MIIC Analysis Plugin

Version 1.0.0

The MIIC Analysis Plugin interface allows users to develop new data analysis routines written in Java. Plugins are developed independantly from the MIIC web application but may run on the web server in the context of a executing a users' inter-calibration plan (ICPlan). Users create an analysis task for their ICPlan by selecting one of the available analysis plugins. Analysis tasks always run after the data for a plan has been collected. Tasks associated with an ICPlan will automatically run whenever the plan has new data available.

Here are the key features and restrictions of MIIC plugin-based analysis:

- Analysis tasks are tied to one ICPlan. They may not view data from other ICPlans.
- Analysis tasks may be "chained" together. Each analysis task that runs has the ability to access all the ICPlan data, plus the results of tasks that ran before it.
- Analysis tasks may accept user input. The input format must be defined by the plugin.
- Analysis task execution is optimized by grouping tasks by plugin. This allows plugins to reduce file I/O. For example, when generating statistics for large plans it makes sense to each data file once and then generate all statistics, as opposed to re-loading the data file for every statistic.

Plugins access ICPlan data and provide analysis results in the formats defined by Abstract Interfaces for Data Analysis (AIDA). AIDA is a set of defined interfaces and formats for representing common data analysis objects. The most important of these interfaces are the Histogram, Profile and Tuple. Please see http://aida.freehep.org for a detailed description of these objects and their interfaces and capabilities.

Accessing ICPlan Data

Plugins access ICPlan data via ITuple and IProfile AIDA analysis objects. ITuples represent filtered data and IProfiles represent averaged data. These formats should be more convenient to plugin authors than using the the NetCDF API, although that is still a possibility in future versions.

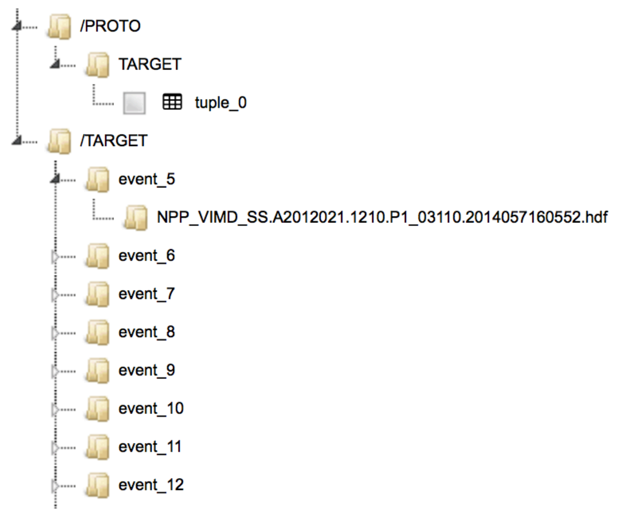
ITuple and IProfile objects are generated by loading the ICPlans' NetCDF data files and converting, on-demand.

ICPlans Collecting Filtered Data

For ICPlans collecting filtered data (a.k.a N_TUPLE format), analysis data is represented using the AIDA structure "ITuple". There will be a separate ITuple for every file queried. ITuples have a "flat" (1D) view of your filtered data.

Inside each ITuple is a column per data variable and a row per observation.

The following shows the analysis objects generated for a ICPlan returning filtered data, in tree form.



```
/PROTO
    TARGET
        tuple_0
/TARGET
    event_5
        NPP_VIMD_SS.A2012021.1210.P1_03110.2014057160552.hdf
    event_6
    event_7
    event_8
    event_9
    event_10
    event_11
    event_12
```

The **PROTO** folder contains prototype JAIDA objects that represent *what* data was collected. This folder will contain one or more ITuple prototypes. Prototypes have a column for every variable stored in the tuple, telling you its name and type, and attributes telling you the min/max of each variable. There are no rows in the ITuple, i.e. no data.

TARGET and **REFERENCE** folders shows you all the data that was collected by events index and source file. In this example, the first 5 events (event_0 to event_4) are missing because no OPeNDAP servers had the required files.

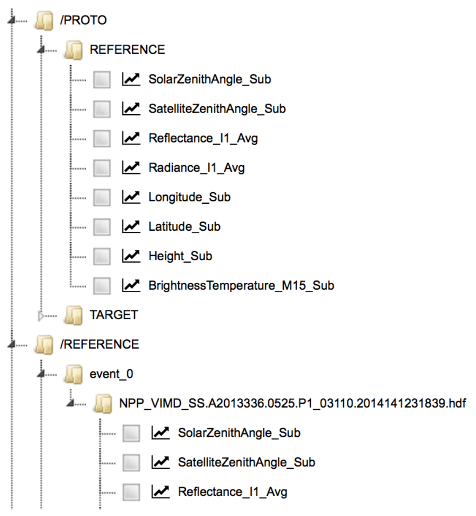
Where are the ITuples?
Note that there are no actual ITuple objects with data stored in this tree. This is because ITuples can be quite large and won't all fit into memory at once.

Instead, plugins must use an interface to retrieve objects on-demand.

ICPlans Collecting Averaged Data

For ICPlans collecting averaged data (a.k.a 2D_HISTOGRAMS), analysis data is represented using the AIDA structure "IProfile2D". Each variable collected will have a separate IProfile2D object per event. IProfile2D objects have 2D binned data and statistics.

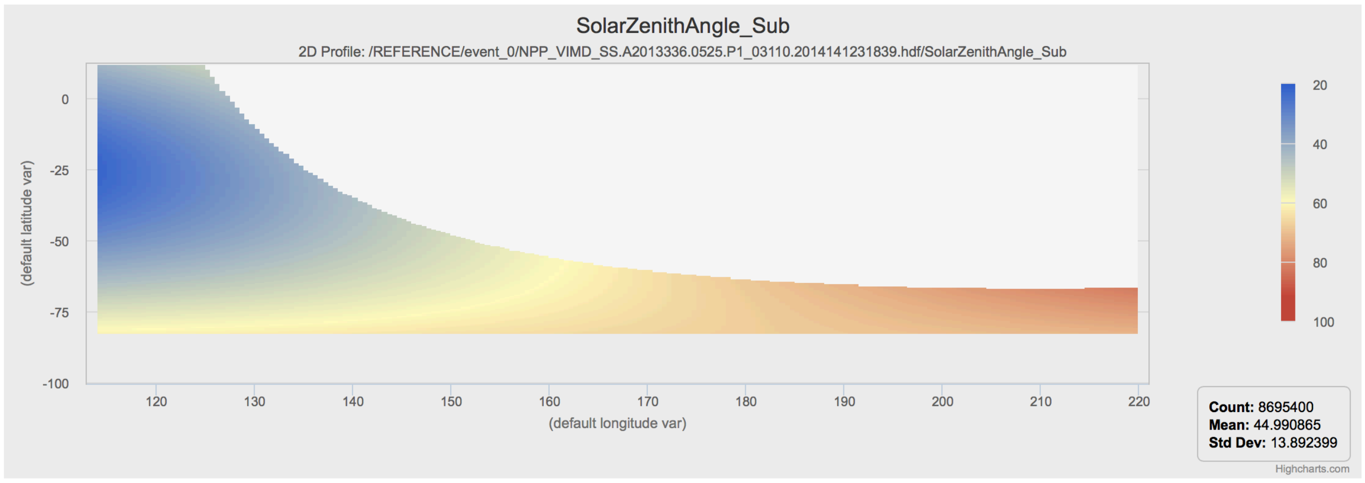
Below is a sample data tree for an ICPlan that collected 2D_HISTOGRAM data:



```
/PROTO
    REFERENCE
        SolarZenithAngle_Sub
        SatelliteZenithAngle_Sub
        Reflectance_I1_Avg
        Radiance_I1_Avg
        Longitude_Sub
        Latitude_Sub
        Height_Sub
        BrightnessTemperature_M15_Sub
    TARGET
/REFERENCE
    event_0
        NPP_VIMD_SS.A2013336.0525.P1_03110.2014141231839.hdf
            SolarZenithAngle_Sub
            SatelliteZenithAngle_Sub
            Reflectance_I1_Avg
```

Here, the **PROTO** folder contains empty 2D Profiles representing the data variables collected by the plan.

As before, the **TARGET** and **REFERENCE** folders show you data collected by event index and file. Because IProfile2D objects are small, they can be kept in memory. IProfile2D objects can be viewed directly on the analysis webpage:



Generating Analysis Results

Analysis plugins must store their results in a JAIDA ITree object. The result itself can be any JAIDA object, and annotations can be used to include additional information. Plugins have read-write access to the results tree, so they may also view and/or modify the output of other plugins.

Analysis tasks, when executed, generate outputs objects and store them in the ITree at well-known locations (i.e. paths like in a filesystem). Plugins are responsible for determining where in the tree to store results, and for ensuring that analysis tasks don't "clobber" or overwrite the results of other analysis tasks.

Statistics plugin ITree layout
The statistics plugin generates statistics objects: IProfiles and IHistograms. It uses a naming convention where the path represents the type of the statistic and the data variable options:

For 1D and 2D Profiles:

- `/ [1D Profile or 2D Profile] / [x (+ y) axis name] / [profiled variable name]`

For 1D and 2D Histograms:

- `/ [1D Histogram or 2D Histogram] / [ x (+ y) axis name ]`

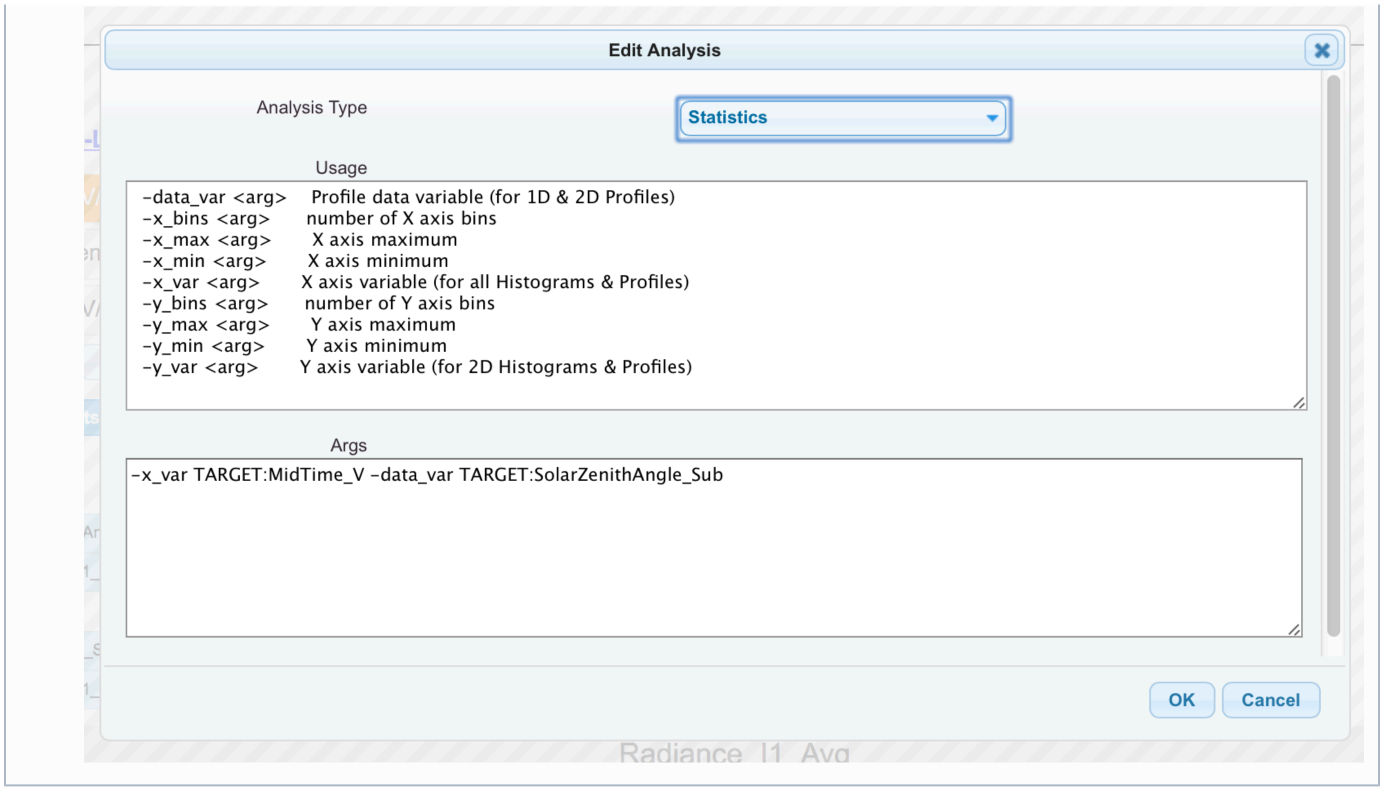
Two statistics of the same type and using the same variables but with different options (e.g. bin size) will "clobber" – i.e. write to the same location in the tree. This may be addressed in a future release.

MIIC Integration

Deployed plugins are available from the MIIC web application and the MIIC REST interface.

The analysis web page allows users to create new analysis tasks by selecting from plugins that support your plan. Users enter arguments, as specified by the plugin. All the information typed in the "args" text area uses autocomplete, so users don't need to remember complex variable names.

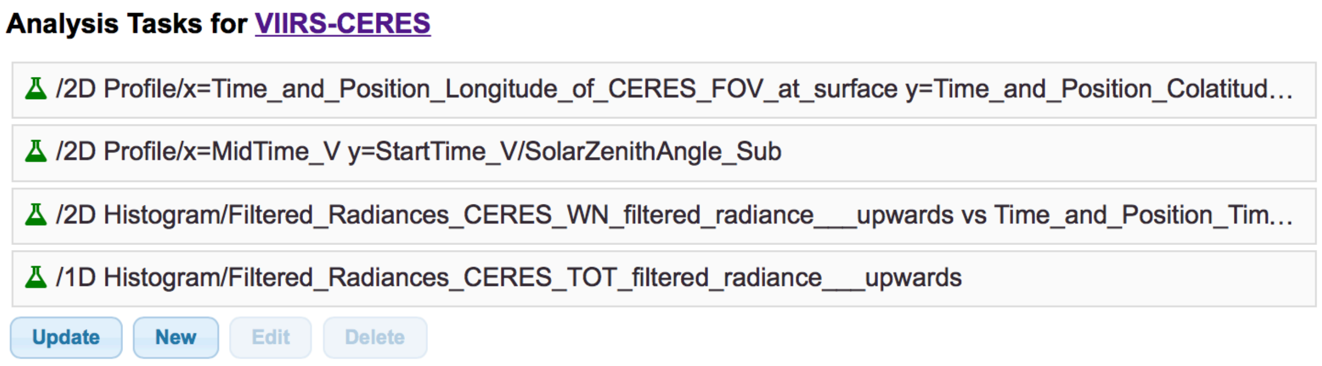
The statistics plugin defines inputs for x & y bins and x, y & data variables as shown below:



```
-data_var <arg>    Profile data variable (for 1D & 2D Profiles)
-x_bins <arg>      number of X axis bins
-x_max <arg>       X axis maximum
-x_min <arg>       X axis minimum
-x_var <arg>       X axis variable (for all Histograms & Profiles)
-y_bins <arg>      number of Y axis bins
-y_max <arg>       Y axis maximum
-y_min <arg>       Y axis minimum
-y_var <arg>       Y axis variable (for 2D Histograms & Profiles)
```

```
-x_var TARGET:MidTime_V -data_var TARGET:SolarZenithAngle_Sub
```

Once created, analysis tasks are named according to the names of result objects generated by the plugin:



Analysis Tasks for VIIRS-CERES

- /2D Profile/x=Time_and_Position_Longitude_of_CERES_FOV_at_surface y=Time_and_Position_Colatitud…
- /2D Profile/x=MidTime_V y=StartTime_V/SolarZenithAngle_Sub
- /2D Histogram/Filtered_Radiances_CERES_WN_filtered_radiance___upwards vs Time_and_Position_Tim…
- /1D Histogram/Filtered_Radiances_CERES_TOT_filtered_radiance___upwards

Developing Plugins

The AnalysisPlugin Interface

All analysis plugins must implement the interface ***gov.nasa.miic.plugin.AnalysisPlugin.***

The interface has methods for:

1. Returning plugin version information
2. Testing plugin ICPlan support
3. Returning plugin Input/Output metadata
4. Performing analysis

Versioning Methods

These methods must return the plugin name and version. This allows us the possibility of adding new plugin versions without breaking ICPlans that rely on older versions:

```
@Override
    public String getName() {
        return "My Plugin";
    }

    @Override
    public String getVersion() {
        return "0.1";
    }
```

Methods to Test ICPlan Support

Plugins must indicate if they work with an ICPlan.

Here is a plugin that refuses to operate on an ICPlan with fewer than a bakers' dozen events:

```
@Override
    public boolean supports(AnalysisData ad) {
        return (ad.getNumEvents() > 12);
    }
```

AnalysisData is the object that provides access to the ICPlan as AIDA analysis objects and is described below.

Input/Output Metadata Methods

Plugins must report what user inputs they accept. Inputs are specified using the Apache Commons CLI library, using the *Options* object.

```
/**
     * Called by MIIC to get the options used by this plugin.
     * Use options for anything that is not in the ICPlan itself.
     * @param data access to the data in an ICPlan using the AIDA API
     * @return Options object representing all required and optional input
options.
     */
    public Options getInputOptions(AnalysisData data);
```

Apache Commons CLI is typically used to build command-line programs that accept complex command-line inputs. In our case, it provides a convenient way for plugins to specify their inputs including:

- Options with one or more argument values (where each option may itself be mandatory or optional)
- Option values of a specific type (integer, etc.)
- Mutually exclusive option groups (i.e. user must choose one among several options)

The plugin must also report what object(s) they will generate or modify given a set of inputs. The *CommandLine* object stores user inputs according to the plugins *Options* object. These inputs are guaranteed to be valid – meaning that options marked as required are provided, and option values are of the correct type.

If the plugin knows at this time that it cannot generate results for the given inputs, it may throw a PluginException.

```
/**
     * Get the results path: where in the ITree the results will be stored
     * @param inputs the inputs are processed somehow to determine the path
     * @throws PluginException if illegal input
     */
    public String getResultsPath(CommandLine inputs) throws
PluginException;
```

Analysis Methods

The analyze function is called by MIIC when analysis must be performed. All ICPlan information is available from the *AnalysisData* object, described below.

The first form supports grouping multiple tasks from the same plugin together. This may allow plugins to organize their processing task so as to minimize the number of times data from the ICPlan must be read from disk:

```
/**
     * Called to perform multiple analyses for one plugin. Grouped together
for performance.
     *
     * Analysis results are attached to an ITree obtained from
AnalysisData.
     * The convention is to store results in a folder with the plugin name.
     *
     * @param data access to ICPlan data
     * @param inputs multiple inputs for the plugin
     * @throws PluginException if the plugin can't generate results
     */
    public void analyze( AnalysisData data, List<CommandLine> inputs )
throws PluginException;
```

The second form is called to run a single analysis task in isolation:

```
/**
     * Called to perform one analysis task.
     *
     * Analysis results are attached to an ITree obtained from
AnalysisData.
     * The convention is to store results in a folder with the plugin name.
     *
     * @param data access to ICPlan data
     * @param input stores input vals for the plugin
     * @throws PluginException if the plugin can't generate results
     */
    public void analyze( AnalysisData data, CommandLine input ) throws
PluginException;
```

Upon failure to generate results a PluginException may be thrown. Otherwise, AIDA object(s) are expected to be stored in the AIDA results tree at the location(s) reported by getResultsPath. Plugins may generate Tuple, Histogram, Profile, DataPointSet, and Function objects. All objects may also provide arbitrary string annotations, which can convey additional meaningful information.

AnalysisData Object

This object provides read-only access to the ICPlan and all of its available raw data for processing. We do not directly expose the ICPlan object to the plugin as this would create an unnecessary software safety risk.

The AnalysisData object is used to:

- Access general ICPlan metadata
- Access the AIDA tree for storing result objects
- Obtain AIDA objects representing ICPlan data (ITuples for filtered data and IProfiles for averaged data)

ICPlan Metadata Access

The bulk of the *AnalysisData* interface allows access to general ICPlan metadata:

```
/**
     * Get the UTC time at the beginning of the intercalibration plan
     * @return UTC time
     */
    public DateTime getBegin();
    /**
     * Get the UTC time at the end of the intercalibration plan
     * @return UTC time
     */
    public DateTime getEnd();

    //
    // event parameters
    //

    /**
     * Get the number of events in the plan
     * @return
     */
    public int getNumEvents();
    /**
     * Get the UTC time at the start of an event
     * @param eventID which event
     * @return UTC time
     */
    public DateTime getEventBeginTime( int eventID );
    /**
     * Get the UTC time at the end of an event
     * @param eventID which event
     * @return UTC time
     */
    public DateTime getEventEndTime( int eventID );
    /**
     * Get northerly bounds of the event region
     * @param eventID which event
     * @return degrees latitude
     */
    public double getEventLatNorth( int eventID );
    /**
     * Get southerly bounds of the event region
     * @param eventID which event
     * @return degrees latitude
     */
    public double getEventLatSouth( int eventID );
    /**
     * Get westerly bounds of the event region
     * @param eventID which event
     * @return degrees longitude
     */
    public double getEventLonWest( int eventID );
    /**
     * Get easterly bounds of the event region
     * @param eventID which event
     * @return degrees longitude
     */
    public double getEventLonEast( int eventID );

    //
    // collection metadata
    //

    public enum CollectionType { TARGET, REFERENCE };

    /**
     * Check if the plan has data of the indicated type
     * @param type
     * @return true if has data
     */
    public boolean hasCollection( CollectionType type );

    /**
     * Get the name of the data collection
     * @param type
     * @return product name
     */
    public String getProduct( CollectionType type );

    /**
     * Get the instrument name for the collection
     * @param type
     * @return instrument name
     */
    public String getInstrument( CollectionType type );

    /**
     * Get the satellite name for the collection
     * @param type
     * @return satellite name
     */
```

```
public String getSatellite( CollectionType type );

/**
 * Get the list of variables available for the collection
 * @param type
 * @return list of variable names
 */
public Set<String> getDataVariables( CollectionType type );

//
// query variable metadata
//

/**
 * Get the minimum legal value for a variable, if known
 * @param type
 * @param var variable name
 * @return minimum value or null
 */
public Double getVariableMin( CollectionType type, String var );

/**
 * Get the maximum legal value for a variable, if known
 * @param type
 * @param var variable name
 * @return maximum value or null
 */
public Double getVariableMax( CollectionType type, String var );

/**
 * Get the single missing value for a variable, if known
 * @param type
 * @param var variable name
 * @return missing value or null
 */
public Double getVariableMissing( CollectionType type, String var );

/**
 * Get the units for a variable, if known
 * @param type
 * @param var variable name
 * @return units name or null
```

```
     */
    public String getVariableUnits( CollectionType type, String var );
```

ITree Access

The *AnalysisData* object will return JAIDA data and results trees. The results tree is where your plugin must store analysis result objects. The data tree is a read-only tree and will be of limited use to clients.

```
public ITree getTree();
public ITree getResultsTree();
```

The syntax and use of JAIDA is outside of the scope of this document. A good first step would be to look at how the Statistics plugin works.

Obtaining ICPlan Data

ICPlan data must be requested by event index, collection, and variable(s) desired. For ICPlans that collect filtered data, ITuple objects will be returned. For ICPlans that collect averaged data, IProfile objects will be returned: IProfile1D (for 1D histogram averaged data) or IProfile2D (for 2D histogram averaged data)

The two functions are functionally equivalent. The first version returns a collection of IManagedObjects, each of which must be cast to the correct type: ITuple, IProfile2D, or IProfile1D. The second version can be used by clients that already know what type of object will be returned.

```
/**
     * Get ICPlan data
     *
     * @param collection TARGET or REFERENCE collection
     * @param event which event
     * @param vars data vars from the collection to retrieve
     * @return collection of analysis object(s) -- ITuple, IProfile1D or
IProfile2D object(s) currently
     * @throws MIICException object could not be generated
     */
    public Collection<IManagedObject> getAnalysisObjects( CollectionType
collection, int event, Collection<String> vars ) throws MIICException;

    /**
     * Get ICPlan data by type
     *
     * @param type provides the Java type of object to return, IProfile1D,
IProfile2D or ITuple
     * @param collection TARGET or REFERENCE collection
     * @param event which event
     * @param vars data vars to retrieve
     * @return collection of analysis object(s) -- ITuple or IProfile2D
object(s) currently
     * @throws MIICException object could not be generated
     */
    public <T> Collection<T> getAnalysisObjects( T type, CollectionType
collection, int event, Collection<String> vars ) throws MIICException;
```

All data is loaded from the ICPlans' NetCDF files, on demand. For IProfiles, there will be one object returned per variable requested. ITuples may contain **all** variables that share a common spatial definition.

ITuple objects are potentially much larger than IProfile objects. IProfiles are loaded from NetCDF disk files once and stored in-memory, whereas IProfiles must be loaded from disk each time. This has practical implications for plugin authors:

- For performance, try to minimize the number of times *getAnalysisObjects* is called for IProfiles. This will reduce the number of disk reads.
- To avoid memory errors, don't keep references to multiple ITuple objects.

ITuple-Specific Functions

To help plugins optimize how their use of ITuples, there is a separate method to obtain a "prototype" tuple definition:

```
/**
     * A ITuple "prototype" tells you what variables are in a tuple, and
what the min & max values for each variable are.
     * This can be used by plugins to prevent calling "getAnalysisObjects"
multiple times (an expensive operation)
     *
     * @param collection
     * @param var
     * @return
     */
    public ITuple getTuplePrototypeByVariable(CollectionType collection,
String var);
```

The ITuple prototype has no data inside it but it is used to inform plugins:

- Which variables are stored in which tuples. (Note that only variables with the same spatial definition will be located in the same tuple)
- Get the min and max values for data variables without having to read all the data

Testing Plugins

The application ***AnalysisPluginDriver*** can be used to test analysis plugins from outside the MIIC web application. This is required so we can ensure plugins behave properly before taking the risk of adding them to the MIIC web application. It may also be useful for clients that prefer to run analysis tasks locally.

The driver loads the plugin, downloads the ICPlan and data from a MIIC server, and performs analysis:

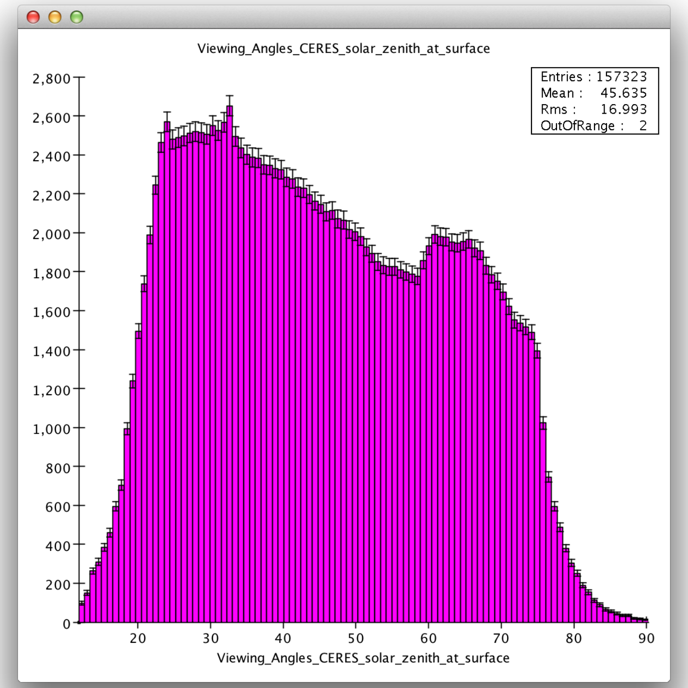
```
usage: AnalysisPluginDriver
 -miic <miic>                miic URL
 -planID <planID>            integer ID of your plan
 -pluginclass <classname>    classname of the analysis plugin (must be in
                             your CLASSPATH)
 -useOldData                 re-use already downloaded ICPlan data if
                             avaialable (doesn't currently check if
                             out-of-date or not)
```

Additional plugin options may be included after the driver options. The driver will report an error here if the supplied plugin options are incorrect.

For example:

```
Loaded plugin: Statistics version: 1.0
org.apache.commons.cli.MissingOptionException: Missing required option:
x_var
 at org.apache.commons.cli.Parser.checkRequiredOptions(Parser.java:299)
 at org.apache.commons.cli.Parser.parse(Parser.java:231)
 at org.apache.commons.cli.Parser.parse(Parser.java:85)
 at
gov.nasa.miic.plugin.AnalysisPluginDriver.main(AnalysisPluginDriver.java:1
39)
usage: Statistics
 -data_max <arg>   Data axis maximum
 -data_min <arg>   Data axis minimum
 -data_var <arg>   Profile data variable (for 1D & 2D Profiles)
 -x_bins <arg>     number of X axis bins
 -x_max <arg>      X axis maximum
 -x_min <arg>      X axis minimum
 -x_var <arg>      X axis variable (for all Histograms & Profiles)
 -y_bins <arg>     number of Y axis bins
 -y_max <arg>      Y axis maximum
 -y_min <arg>      Y axis minimum
 -y_var <arg>      Y axis variable (for 2D Histograms & Profiles)
```

By default, the plugin driver will attempt to render any objects you have added to the results tree. For example, the default rendering of a 1D histogram object:



Deploying Plugins

To make analysis plugins available to MIIC clients, they must be packaged and deployed to a MIIC server.

Plugin packaging is a jar file containing plugin class files and XML configuration file(s). Configuration files are used by the server to discover and configure plugin(s). The file name is unimportant but it must be located in the folder "META-INF/plugins".

The format of the configuration file is springframework "bean" syntax. It simply defines a bean (i.e. Java object) for your plugin from your class that implements the *AnalysisPlugin* interface. It then passes this object to the *analysisPluginRegistry* using a utility called *PluginBeanFactoryPostProc essor*.

Below is the XML configuration to configure a the MIIC plugins jar. This jar contains two plugins: *Statistics* and *SumBinnedData*:

META-INF/plugins/miic.plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="
   http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
   http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.1.xsd">

 <!--  The plugin objects (may themseleves use spring config if desired)
-->
 <bean id="stats" class="gov.nasa.miic.analysis.plugin.Statistics"/>

 <!-- The factory bean adds plugins to a list so the system can find them
-->
 <bean class="gov.nasa.miic.util.PluginBeanFactoryPostProcessor">
  <property name="extensionBeanName" value="analysisPluginRegistry"/>
  <property name="pluginBeanName" value="stats"/>
 </bean>

 <!--  The plugin objects (may themseleves use spring config if desired)
-->
 <bean id="sumbinned" class="gov.nasa.miic.analysis.plugin.SumBinnedData"/>

 <!-- The factory bean adds plugins to a list so the system can find them
-->
 <bean class="gov.nasa.miic.util.PluginBeanFactoryPostProcessor">
  <property name="extensionBeanName" value="analysisPluginRegistry"/>
  <property name="pluginBeanName" value="sumbinned"/>
 </bean>
</beans>
```

Once packaged in a jar, the jar must be copied to